**ANALYSIS OF ALGORITHMS**

**HALF TRAVELER SALESMAN PROJECT**

**Content Table:**

How The Program Works …2-3

Work Division…………………3

Output…………………………..4

**How The Program Works**

We have a City class that holds the id, x, and y variables.

Firstly, the cities in the txt file are obtained and created new City objects.  
The objects that are created are stored in a List<City>.

Secondly, we had the need to calculate every distance between all cities. However, since creating a distance matrix would cause significant memory usage, we preferred to calculate distances through the implementation without a matrix. With this decision, we also saved time by avoiding n² distance calculation.

**Note**: The distance calculation is done by the formula given in the project file.

A square root of a mathematical equation

Description automatically generated with medium confidence

As the next step, we start analyzing which half of the cities to consider as our tour.  
For this purpose, we used a greedy approach to the problem. We chose our start city as the city in the index 0 and store it in bestTour array.

Starting from that city, we choose the next city by finding the closest city to the current city and taking the closest city in our bestTour array, then it continues doing the same operation until the number of the bestTour array reaches ⌈half of the cities⌉.

**Note**: To have fast access, we used arrays to keep the cities’ id while we were finding the best tour and doing searches among the cities.

By now, we obtained half of the cities that will be on our tour. However, to reach a more optimal result, we used the 2-opt algorithm which is a local search heuristic algorithm that iteratively swaps pairs of edges in the tour.   
We observed that the tour length is drastically shortened after implementing that algorithm.

How the 2-opt algorithm works in our program is that the algorithm iterates through all pairs of edges in the tour and evaluates the potential improvement by swapping them. It calculates the new tour length by subtracting the lengths of the original edges and adding the lengths of the swapped edges using the distance calculation. If the new tour length is shorter than the current length, the edges are swapped. The algorithm continues exploring other potential swaps until no further improvement can be done.

Throughout the iterations, when a better tour is found the bestTour array is copied from the improved tour array.

With this step, we obtained the best half-tour in the given cities.

Then, the total distance to travel in the tour including returning to the start city is calculated and stored in the output String with the cities in the tour.

Finally, the output is written in output files.

We used other algorithms as well to obtain optimal results such as Ant Colony Optimization and 3-opt algorithms. However, we considered that the running time of the program is as significant as the optimality of the result, so we chose to apply 2-opt algorithms to our program.

As a result, we aimed for fast and optimal results by applying the algorithms explained above.

**Output**

**A picture containing text, screenshot, font, design

Description automatically generated**